# Downsizing

## EMBEDDED TECHNIQUES

# Atmel's AT89C2051 Flash-based Microcontroller

John Dybowski

This little chip is essentially a flash-based 8051 with some extra RAM in a 20-pin package. True, it does possess fewer I/O pins, but it provides all the peripherals and functions of a standard 8051.

It's funny the way things work out. I would never have imagined that someday I'd be a proponent of the 8051 architecture. But, in light of the work I've been doing lately with a variety of 8051 derivatives, this must be exactly the way I'm coming across. Quite frankly, I'm not entirely comfortable with the situation.

I still remember my first encounter with the 8051. I immediately recognized its utility as a Boolean processor and was quite impressed with the ease with which it handled bit-oriented I/O. At the same time, I was appalled by the seemingly random architecture and the absence of some rather fundamental instructions. These idiosyncrasies continue to irk me and appear all the more enigmatic in light of recent microcontroller advances.

Still, it pays to remember that silicon was a much more precious commodity at the time the 8051 was developed than it is today. Obviously, the seemingly arbitrary exceptions and bizarre architectural quirks were the result of gut-wrenching decisions made to bring the processor's principal features in line with a constrained silicon budget. It's not an uncommon problem. I'm sure the consequences of those unpleasant decisions were not taken lightly.

Although the mechanics of crafting silicon have changed, the fundamental aspects of doing business remain the same. And, this is the key to the puzzle since this is, after all, a business venture and not a science fair project. As a case in point, consider how this same scenario has replayed over the years.

More recently, the same tough choice between price, performance, and features was played out with yet another strange, but extremely successful microcontroller—the much maligned, Microchip PIC16C54. To me, this processor is the epitome of business sense winning out over technological constraints.

The thing doesn't even have an interrupt. Although technically a no-brainer, adding an interrupt would have resulted in totally blowing the target price. Talk about tough choices. As it turned out, the chip's many deficiencies were each countered by a combination of skillful marketing and imaginative technical support. They even convinced a sufficiently large number of engineers that they really didn't need that interrupt after all!

Both the 8051 and the 16C54 architectures have established themselves in their respective spheres of application. As Microchip develops the PIC series into larger, more capable devices, it's inevitable that the architecture's basic simplicity has become somewhat obscure.

Ironically, the 16C54's big brothers are starting to look a lot like 8031s, a rather questionable distinction. It's also a dubious arena to compete in since it is well-served by some firmly established controllers. To me, the original 16C54 is still the soul of the PIC family. Warts and all, this is the part I turn to when I need to implement bare-bones functionality and all I can afford is a $2 chip.

### WRAP IT UP

Formerly, using a higher-level language on 8051-class processors was considered a disputable practice at best. After all, the basic architecture has little enough real stack space to begin with and no stack manipulation capability at all—obviously, a major drawback with a stack-oriented language. Combining these handicaps with a meager instruction set and all those overlapping code, data, and bit segments, it's no wonder that many viewed using code generators on such micros as nothing more than an interesting diversion with little practical consequence.

Evidently, in spite of these obstacles, compiler developers have managed to prevail. Actually, it could be argued that it is because of these obstacles that they have been so successful. The fact is, placing a good code generator between such an architecture and yourself isn't that bad an idea. When I'm writing code, I don't mind dropping to the machine level when it's necessary or expeditious. I'm also well aware of how counterproductive it is to remain at that level any longer than necessary.

An analyses of the situation reveals that some of the most popular embedded-C compiler implementations run on some of the more "difficult" processors. This could be construed as something of a paradox. The implication seems to be that these popular, although irregular, processors assume a level of anonymity through the insulating effect of the language compilers.

From this, it follows that selecting an appropriate architecture should be a relatively straightforward and rational exercise. This might be true if a typical embedded program could be coded entirely using a higher-level language. That this is not the case, especially with 8-bit processors, is evident by the continued popularity of older architectures like the ones I've been describing—so much for rational thinking. If it looks to you like a case of the tail wagging the dog, then you're not alone.

The fact is, as any embedded developer well understands, portable code is something that is extremely elusive and difficult to accomplish in most embedded applications. Usually, system I/O turns out to be such a big part of the overall picture that it is the primary concern.

And, regardless of "proper" coding techniques, embedded programs are often laced with timing dependencies that rely on the execution times of various pieces of code. When doing this type of programming, the bottom line is that you have to know when to get down to the machine level and you must have a good understanding of the quality of code your compiler is generating.

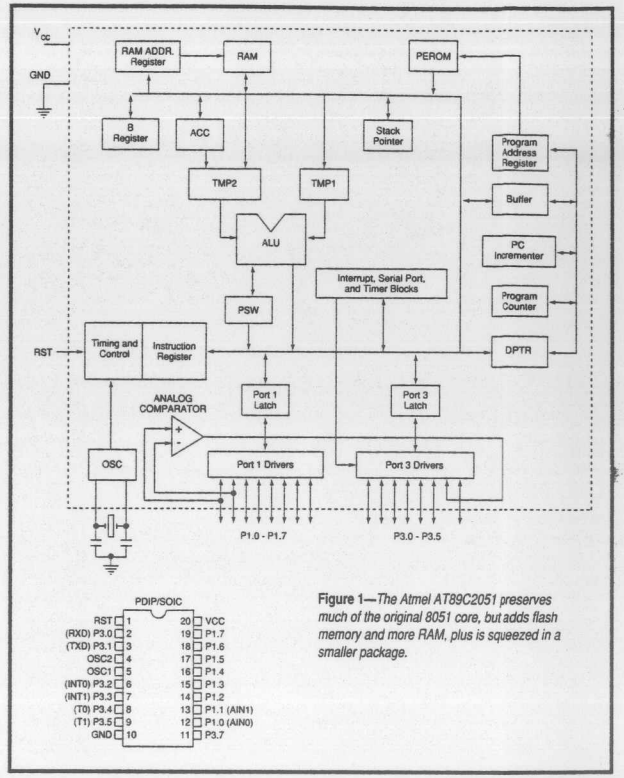


**Figure 1**—*The Atmel AT89C2051 preserves much of the original 8051 core, but adds flash memory and more RAM, plus is squeezed in a smaller package.*

Still, in spite of all the complicating issues, the productivity gains that can be realized from using a higher-level language offer a compelling incentive to rethink the way you've been writing your embedded programs.

The benefits of using language compilers can be perceived differently depending on your particular needs. Minimally, an efficient compiler can function as a sort of stylized assembler. This is especially true of the C programming language which affords excellent control of low-level functions. The incentive for this category of usage is pretty strong with many of the more limited controllers and processors.

Recently, some of the most difficult C implementations have been realized on resource-starved controllers such as the PIC16C54. Frankly, when I first found out that engineers were actually seeking C compilers for the 16C54, I was a bit baffled. I mean, with a two-level stack, 512 words of program memory, and a few dozen bytes of read/write storage, it didn't exactly seem like a good fit. It looked like the compiler's overhead alone could swamp the processor's meager resources.

It turns out that there are some very smart software people and now there are a number of C compilers available for these minuscule processors. Reports from the field indicate that this stuff really works. I haven't tried one yet but, having done a couple of projects involving the 16C54, I'm very interested. You see, one of the big attractions of a language compiler for

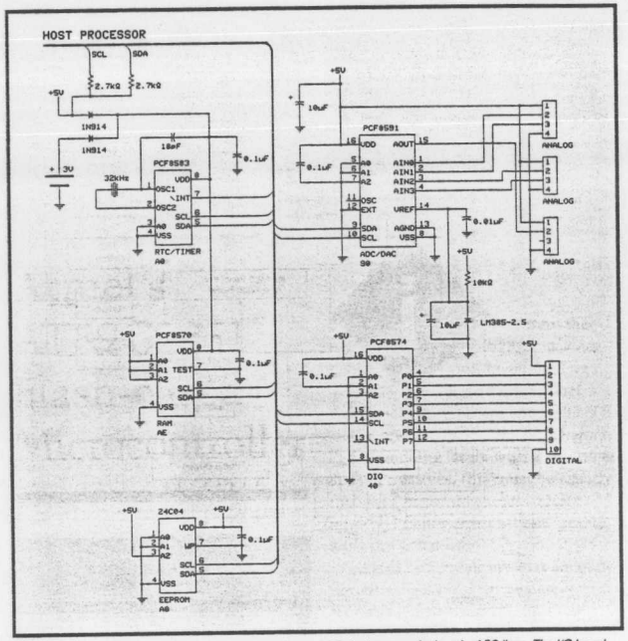


**Figure 2a**—*Host I/O bits are preserved by connecting most I/O devices to a single pair of I²C lines. The I/O board includes a real-time clock and timer, RAM, EEPROM, analog in, analog out, and digital I/O, all connected to just two wires.*

me is its ability to handle many of the tedious programming details. And, there are many of these indeed when working with very small processors.

Put another way—if it's ugly, wrap it up!

## LISTEN TO YOUR CUSTOMERS

Recent evolutionary trends have involved enhancing "standard" processor architectures. For better or worse, the 8051 is now perceived as standard indeed. In recent columns, I've shown how this basic architecture has gained in performance and capability while retaining its standard footprint and pinning.

There's also been a trend to downsize the architecture. Downsizing has led to a loss of some capabilities, a reorganization of the fundamental processing core, and a reduction of on-chip memory. The high end has been well served by a variety of capable derivatives. The low end until recently has been...well, the low end. But finally, someone has introduced a very small 8051 that doesn't give up any of its fundamental features.

The new, flash-based AT89C2051 from Atmel is a welcome addition to the 8051 family. More than just another scaled down 8051, this processor is a full-blown 8051 housed in a 20-pin package. All the standard 8051 peripherals have been retained including two 16-bit timers along with their gating controls and associated interrupts, two external interrupts, and a full-duplex hardware UART. This is in addition to 2 KB of flash program memory and the standard 128 bytes of internal RAM.

It may not seem like a big deal at first, but it is the inclusion of the full 128 bytes of internal RAM that opens up substantial opportunities for using this part with standard commercial C compilers. While using a compiled language is certainly an option with some of the more restricted 8051 derivatives, things naturally get quite limited with the 64 bytes of RAM most of these smaller chips offer. And, if you're worried about code efficiency, the AT89C2051 specifies an operating frequency from DC to 24 MHz. This gives you the elbow room to handle any reduction of efficiency that results from using a code compiler.

Another area that competitive products have consistently ignored is the inclusion of a hardware UART. Although many of the competing parts offer hardware-assisted I²C interfaces, it's obviously of little consolation when you have to provide for standard asynchronous communications.

From where I sit, the outcry for a hardware UART has been long, loud, and clear. Needless to say, thus far, the semiconductor manufacturers have been totally unreceptive to what users have been calling for. Their response to these requests has simply been to either say that you really don't need it or to suggest running one entirely in firmware. I'm surprised this "what we've got is what you need" mentality has prevailed as long as it has.

As a result of this attitude, manufacturers have left themselves wide open, jeopardizing a significant market share. If the AT89C2051 offered nothing more than just a hardware UART, its success would be ensured. The fact is it's got a lot more than that going for it. Figure 1 shows the contents of this 20-pin processor in block form.

## I LIKE I²C

The AT89C2051 possesses a full 8051 architecture, it just happens to have fewer I/O pins. Obviously, to design a general-purpose embedded computer around such a device, it's necessary to conserve these I/Os. The most efficient way to hook up a lot of peripherals using just a couple of lines is over the I²C bus.

At first, it might seem that not having any hardware I²C support might prove to be a major liability in what essentially amounts to an I²C-based computer. But, with the exception of some very specialized applications, it is not a problem at all.

Remember that I²C defines a synchronous protocol that is capable of running all the way down to DC. With the AT89C2051's complement of internal and external interrupts, it's easy to arrange time-critical processes to run entirely under interrupts. This lets you bit-bang the I²C and still stay live to service your real-time operations.

Obviously, this is luxury you don't have when bit-widdling asynchronous communications, which by definition must adhere to relatively strict timing constraints. Sure, you can run asynchronous communications in the interrupts, but there are serious limitations to the maximum bit rate you can reach.

Some I²C peripherals come prepackaged and ready to go. The small multifunction card shown in Photo 1 illustrates a number of such useful functions. Figure 2 depicts the card's schematic. The I/O-related functions include four single-ended or two differential 8-bit analog inputs, one 8-bit analog output, and eight digital I/O points.

The card also provides a real-time clock-calendar timer with 256 bytes of RAM, a dedicated 256 bytes of RAM, and 512 bytes of E²PROM. The RTC and RAM are backed using a BR1225 lithium coin cell that is also contained on the card. The card, including connectors, measures only 1" × 3". If your application doesn't need these functions, then simply don't load the card. I²C, especially a firmware-based implementation as used here, carries very little hardware overhead. Even the requisite pull-up resistors are located on the card.

Higher-level I²C functions are realized using a combination of I²C and logic ICs. Using a couple of I²C-to-parallel converters and some support logic, a 4 × 20 LCD and a 4 × 4 keypad can be supported using the same two I/O lines. The control card for this peripheral system attaches almost invisibly beneath the standard 4 × 20 supertwist LCD panel as shown in Photo 2. This arrangement leaves a number of general-purpose I/O lines free to drive a beeper, LEDs, and other indicators.

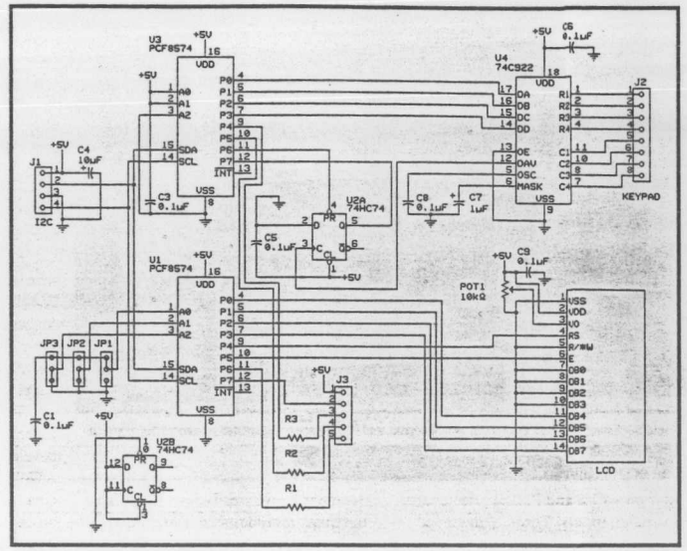


**Figure 2b**—*Another interface board handles a 4 × 20 LCD, 4 × 4 keypad, and general I/O over two wires.*

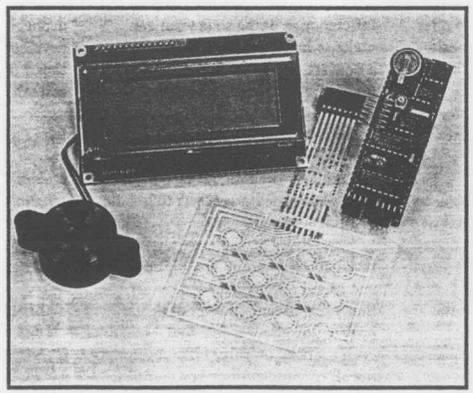

**Photo 1**—*A sample of the peripherals which can be easily hooked up using a combination of I²C and logic ICs. All the devices shown in Figure 2a fit on a 1" × 3" board (upper right) while those in Figure 2b fit under the LCD display.*

## PROCESSOR I/O

So far, the I/O devices I've described encompass all the peripheral functions a very small embedded computer might need. The beauty of this arrangement is that, although the 89C2051 has only 15 I/O pins to begin with, after accommodating all these functions we are still left with 13 for general usage. These I/Os can be directly tested and set by the processor and are therefore very fast. Among these general-purpose I/O bits are two external interrupts, receive-and-transmit connections to a full-duplex UART, and the gating signals for the two internal 16-bit timers.

Although the on-chip I/O lines are for the most part 8051 compatible, there are some important differences. Refer again to Figure 1. You can see the familiar 8051 pinout nomenclature has been preserved. From this figure, you can also see

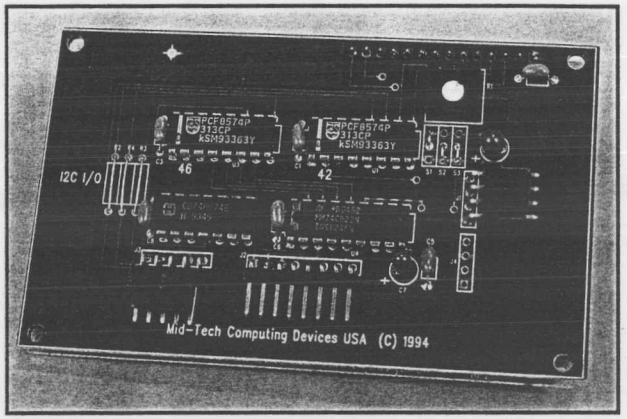


**Photo 2**—*A standard 4 × 20 LCD display, 4 × 4 keypad, and I²C interface are supported on a board that mounts on the back of the display.*

that pins P1.0 and P1.1 are designated as analog inputs. These high-impedance inputs serve as positive (AIN0) and negative (AIN1) inputs to the built-in precision analog comparator. P3.6 is not available externally and instead takes the output of this comparator as its input. To facilitate interfacing to real-world loads without the need for external buffers, ports 1 and 3 are rated to sink a full 20 mA when running at 5 V.

## MAKE SENSE?

If you accept that the peripheral mix I've described for my system is not unusual, then perhaps I can offer some insight from my own experiences on issues like code portability, code reuse, and the enduring success enjoyed by the 8051 architecture. In this context, I will also elaborate on why I feel the 89C2051 is such a significant processor.

Simply put, the I²C peripheral set I've described has served me well in a variety of configurations on a variety of host systems for a number of years. I've developed a fair amount of C code to support these functions, but the main low-level drivers are all, of necessity, written in hand-tweaked assembler.

This in itself represents a significant investment of time and effort. To this, of course, you must add the "standard" overhead of the trade. This overhead includes such things as interfacing low-level code to the compiled code (usually quite different for each compiler you use); debugging, testing, and validating functions; documentation; and continuing support. Underestimating these associated tasks can have a deleterious effect on your continued success in this field!

Electronics means different things to different people. To those of us that have bills to pay, it's a business. In this context, the bottom line revolves around spinning off applications. Efficiency in doing so is what can make or break it. Sure I could code up a new set of low-level drivers for a different processor in a relatively short period (disregarding the overhead of course). And, if I had to, I could take all that C code and rewrite it in assembler if the processor I was using didn't have the horsepower to handle a compiled language.

Individually, these are fairly small tasks. It's only when you look at the overall picture that it becomes evident that a significant amount of time can be spent on these tasks. And, I haven't even touched on things like regenerating all my interrupt-driven communication routines, timer-capture functions, and the like. There comes a point when you have to ask yourself where you want (or can afford) to spend your development time.

## EMBEDDED TOOLS

I'm sure it's obvious I have more than just a passing interest in the new Atmel processor. Next month, I'll cut through the fluff and devote my attention to the design and development of a very small, general-purpose computer based on the AT89C2051.

Having defined the 1" × 3" form factor for the I²C peripheral card, I will use the same footprint for the main processor card. This card will contain the 20-pin, flash-based processor; an RS-232 and RS-485 interface; power supply; and prototype area.

The main focus of my column, however, will be the associated development system for the AT89C2051. In this respect, I'll also be covering the AT89C51 flash-based processor that, with some hardware assistance, emulates the AT89C2051. The system works with a variety of target systems since it uses a standard umbilical cable hookup.

Of course, once you get an application up and running, you'll want to program the AT89C2051 flash memory. This will be accomplished with the built-in flash programmer contained right on the development system. As usual, I'll be calling on my esteemed colleague, Dave Dunfield, for his fine PC-hosted development tools. I'll be using his simulator and remote-debugging software to round out the development system. For code generation, I plan to use his Micro-C and assembler products.

*John Dybowski is an engineer involved in the design and manufacture of embedded controllers and communications equipment with a special focus on portable and battery-operated instruments. He is also owner of Mid-Tech Computing Devices. John may be reached at (203) 684-2442 or at john.dybowski@circellar.com.*

**IRS**

425 Very Useful
426 Moderately Useful
427 Not Useful